Understanding Binary Search

Binary search is an efficient algorithm for finding a target value within a sorted array. It works by repeatedly dividing the search interval in half. If the target value is less than the middle element of the interval, the search continues in the lower half; otherwise, it continues in the upper half. This process repeats until the target value is found or the interval is empty.

How It Applies to Guessing Numbers

In the Questions program, the goal is to guess a number the user is thinking of within a specified range using a series of yes/no questions. Each question narrows down the range of possible numbers.

Key Concept: Powers of Two and Binary Representation

Each question in binary search can be thought of as a binary decision that splits the current range into two halves. This is directly related to the binary representation of numbers and the power of two.

1. **Number of Questions (k)**:
   - Each question allows the algorithm to halve the search space.
   - If you ask k questions, you can halve the range k times.
2. **Number of Possible Values (2^k)**:
   - Halving the range k times means you can distinguish between 2^k different outcomes.
   - This is because each question provides a binary (yes/no) decision, and k binary decisions can represent 2^k different values.

Detailed Explanation

Let's break it down with an example:

Example: $k = 3$ (3 Questions)

- If you have 3 questions, you can distinguish between $2^3 = 8$ different numbers (0 to 7).

Binary Representation

Each yes/no question can be thought of as a binary digit (bit):

- **Yes** (greater than or equal to mid) might represent 1.

- **No** (less than mid) might represent 0.

With 3 questions, you get 3 binary digits, which can represent numbers from 0 to $2^3 - 1 = 7$.

Binary Search in Action

1. **Initial Range**: 0 to 7 (8 values)
2. **First Question**: Is the number greater than or equal to 4?
   - If yes, the number is in the range 4 to 7 (upper half).
   - If no, the number is in the range 0 to 3 (lower half).
3. **Second Question**: Depending on the first answer:
   - If the first answer was yes, ask if the number is greater than or equal to 6 (splitting 4-7).
   - If the first answer was no, ask if the number is greater than or equal to 2 (splitting 0-3).
4. **Third Question**: Further split the range based on the second answer.

After 3 questions, you will have narrowed down the number to a single value.

General Case

- With k questions, each question splits the range in half.
- This means k questions create 2^k possible combinations of yes/no answers, which can represent 2^k different numbers.

Here's how you can implement the `Questions` class to take the maximum number n as a command-line argument and perform a binary search to guess the user's number.

Implementation

```java
import lib.StdIn;
import lib.StdOut;

public class Questions {
    // Recursive binary search to find a number in the range [lo,
hi)
    public static int binarySearch(int lo, int hi) {
        // Base case: only one number in the range
        if (hi - lo == 1) {
            return lo;
        }

        // Calculate the midpoint
        int mid = lo + (hi - lo) / 2;
        StdOut.print("Greater than or equal to " + mid + "? (yes /
no) ");
```

```
        // Read the user's response
        if (StdIn.readBoolean()) {
            // If the number is greater than or equal to mid,
search in the range [mid, hi)
            return binarySearch(mid, hi);
        } else {
            // If the number is less than mid, search in the range
[lo, mid)
            return binarySearch(lo, mid);
        }
    }

    public static void main(String[] args) {
        // Number of possible values (n) is taken from command line
arguments
        int n = Integer.parseInt(args[0]);

        // Prompt the user to think of a number in the range [0, n-
1]
        StdOut.print("Think of a number between 0 and " + (n - 1) +
"\n");

        // Perform binary search to guess the number
        int guess = binarySearch(0, n);

        // Print the guessed number
        StdOut.println("Your number is " + guess);
    }
}
```

Explanation

1. **Command-Line Argument**:
   - The program takes a single command-line argument n, which is the maximum number (exclusive).
   - This means the user is thinking of a number between 0 and $n-1$.
2. **Binary Search**:
   - The `binarySearch` method performs a recursive binary search within the range $[lo, hi)$.
   - It keeps narrowing down the range by asking the user if their number is greater than or equal to the midpoint of the current range.
3. **Base Case**:
   - When the range has only one number left (i.e., $hi - lo == 1$), it returns that number.
4. **Recursive Case**:

- It calculates the midpoint mid and asks the user if their number is greater than or equal to mid.
- Depending on the user's response, it recursively calls itself with the updated range.

Proving Correctness

To prove the correctness of the implementation, consider the following points:

1. **Base Case**:
   - When the range $[lo, hi)$ contains only one element, the correct number is found and returned.
2. **Recursive Case**:
   - At each step, the current range is split into two halves by calculating the midpoint.
   - Based on the user's input, the algorithm continues the search in the appropriate half.
   - This ensures that with each question, the size of the search space is halved, leading to the correct number being found eventually.
3. **Range Reduction**:
   - Initially, the range is $[0, n)$.
   - Each question reduces the range by half:
     - If the answer is "yes," the new range is $[mid, hi)$.
     - If the answer is "no," the new range is $[lo, mid)$.
4. **Termination**:
   - The algorithm terminates when the range contains only one number, ensuring the correct number is returned.
5. **Number of Questions**:
   - The maximum number of questions needed is $\lceil \log_2 n \rceil$, which is the height of the binary search tree for n elements.

Example

Let's take an example to see how it works with $n = 8$:

- User thinks of a number between 0 and 7.
- The program starts with the range [0, 8).
- Questions asked:
  1. Is the number greater than or equal to 4? (splits range 0-7 to 0-3 and 4-7)
  2. Based on the answer, the next question narrows it further, e.g., is it greater than or equal to 6? (if in the range 4-7).

3. The final question narrows it to a single number.

This ensures the number is guessed correctly within $\lceil \log_2 8 \rceil = 3$ questions.